ELSEVIER

Theoretical Computer Science 170 (1996) 245–276

Theoretical Computer Science

Using induction and rewriting to verify and complete parameterized specifications

Adel Bouhoula *

INRIA Lorraine and CRIN, Campus Scientifique, 615, rue du Jardin Botanique - B.P. 101, 54602 Villers-lès-Nancy Cedex, France

Received May 1995; revised November 1995
Communicated by M. Wirsing

Abstract

In software engineering there is a growing demand for formal methods for the specification and validation of software systems. The formal development of a system might give rise to many proof obligations. We must prove the completeness of the specification and the validity of some inductive properties. In this framework, many provers have been developed. However they require much user interaction even for simple proof tasks. In this paper, we present new procedures to test sufficient completeness and to prove or disprove inductive properties automatically in parameterized conditional specifications. The method has been implemented in the prover SPIKE. Computer experiments illustrate the improvements in length and structure of proofs, due to parameterization. Moreover, SPIKE offers facilities to check and complete specifications.

Keywords: Theorem proving; Sufficient completeness; Implicit induction; Parameterized conditional specifications; Term rewriting systems

1. Introduction

Algebraic specifications [38] provide a powerful method for the specification of abstract data types in programming languages and software systems. Often, algebraic specifications are built with conditional equations. Semantically, the motivation for this is the existence of initial models; operationally, the motivation is the ability to use term rewriting techniques for computing and automatic prototyping. One of the most important issues within the theory of algebraic specifications is the specification of parameterized data types. Most common data types like *list* are in fact parameterized types, *list*(*data*). The key idea is to consider the parameter part *data* as a formal algebraic specification which can be actualized (i.e. instantiated) by other predefined

* e-mail: bouhoula@loria.fr.

PII S0304-3975(96)00005-9

algebraic specifications like *nat*, *int* or *bool*. Hence, we can obtain from the parameterized specification *list(data)* the three value specifications corresponding to lists of natural numbers, lists of integers and lists of boolean values. The benefit of this process is not only an economy of presentation but also the automatic correctness of all the value specifications, provided that the parameterized specification *list(data)* is correct and the actual instantiation is valid. These are very important properties for building up larger data types and software systems from small pieces in a correct way. Sufficient completeness and consistency are fundamental notions for guaranteeing correctness of a parameterized specification. Also, they are very useful in proofs by induction. Informally, given a conditional specification S and a set of distinguished operators C, called *constructors*, S is said to be sufficiently complete if any normal form of a ground term is a constructor term, i.e. a term built only from constructors. Guttag showed that this property is undecidable [21]. However, some syntactic criteria can be given. Most of them are based on rewriting methods [22, 24, 14, 28, 13, 32]. In the context of conditional parameterized specifications, the art is less developed. This is mostly due to the fact that the problem is much harder. In this paper, we give an effective method for testing this property for parameterized conditional specifications.[1] This method is inspired by [28, 9] and it is based on the notion of *Pattern* trees.

Another direction is to make use of parameterization at the proof level and to develop a generic proof method. This approach allows us to have shorter and more structured proofs. A generic proof for a parameterized specification must be given only once and can be reused for each instantiation of the parameter. We are interested in automating proof by induction. Many tools for proof by induction have been developed for non-parameterized specifications: The first type applies explicit induction arguments on the term structure [1, 10, 12]. The second type involves a proof by consistency [33, 24, 25, 18, 31, 2, 20]. More recently, new methods were developed that do not rely on the completion framework [30, 37, 9, 8].

The inductive theory of a parameterized specifications has been studied by Navarro and Orejas [34]; their results generalize [35]. But they do not give effective methods to prove inductive theorems. Ganzinger [19] considered parametric conditional equational specifications that allow arbitrary first-order formulas as "parameter constraints", but he was interested in ground-confluence results and not in inductive theorem proving. Kirchner [27] has studied proofs by induction in the unconditional case (where the parameter theory is equational) using techniques of proof by consistency. Becker [3] has dealt with proof by consistency in parameterized positive/negative conditional equational specifications. To conclude, most of the work in proof by induction considers only the technique of proof by consistency. It is generally accepted that such techniques may be very inefficient since the completion procedure often diverges. For that reason, we adopt here a method that does not require completion.

[1] To our knowledge, no previous implementation was able to check the sufficient completeness of parameterized conditional specifications.

The system SPIKE[2] [5, 6] has been developed in this framework. It incorporates many optimizations such as powerful simplification techniques. SPIKE has proved several interesting theorems in a completely automatic way [4], that is, without interaction with the user and without ad hoc heuristics. It has also proved the challenging Gilbreath card trick with only 2 easy lemmas which are given in the beginning of the proof [8]. This example was treated by Boyer in NQTHM [10] and Zhang in RRL [39]. Unlike us, they require a lot of lemmas, some of them being non-obvious. To our knowledge, SPIKE is the only one that can disprove *non-trivial* inductive theorems in conditional theories without any interaction. None of the well-known induction provers has been designed to refute false conjectures. For an inexperienced user, a serious weakness of the NQTHM, CLAM [12] and RRL systems is that they do not provide much useful information when they fail. In particular, it is not clear from the generated output whether the conjecture being proved is false or a proof of the conjecture is likely to need additional lemmas. Unlike the latter, SPIKE guarantees when it fails that one of the initial conjectures is not an inductive theorem provided that the axioms are boolean and ground convergent.

We give in this paper a new procedure for proof by induction in parameterized conditional specifications. Our procedure relies on the notion of *test set* which can be seen as a special induction scheme that allows us to refute false conjectures by the construction of a counterexample. Our definition of test set is more general than the previous one given in [8]. It permits us to obtain a smaller test set, which improves efficiency. As in our previous procedure [8], to prove conjectures, we just instantiate them with terms from the test set at induction positions and simplify them by axioms, other conjectures or induction hypotheses. The method does not require any hierarchy between the lemmas. They are all stored in a single list and the use of conjectures for mutual simplification simulates *simultaneous induction*. Unlike our previous method [8], this new procedure, when limited to non-parameterized conditional specifications, can refute general clauses; refutational completeness is also preserved for boolean ground-convergent rewrite systems even if the functions are not sufficiently complete and the constructors are not free. The method has been implemented in the prover SPIKE. Experiments illustrate the improvements in length and structure of proofs, due to parameterization.

The organization of this paper is as follows. In Section 2, we briefly introduce basic concepts about term rewriting. In Section 3, we characterize the inductive theory defined by a parameterized specification. We present in Section 4 the procedure for testing sufficient completeness and we prove its soundness and completeness. We also describe a session with SPIKE to give an idea about the interaction with the user in order to recover a sufficiently complete specification. In Section 5, we define the notions of induction variables and test sets. In Section 6, we give a general inference system to perform induction and to refute false conjectures and we show its soundness. The strategy is proved refutationally complete for conditional equations with boolean

[2] SPIKE is available via anonymous ftp from ftp.loria.fr, in the directory /pub/loria/protheo/softwares/Spike.

preconditions (Section 6.3). Section 7 is dedicated to computer experiments with our SPIKE system. We give a comparison with our previous method for non-parameterized specifications and we show how proofs in parameterized specifications are shorter and more structured (Section 7.1). In Section 7.2, we give a complete example of a refutation of a false conjecture.

2. Basic concepts

We assume that the reader is familiar with the basic concepts of term rewriting, equational reasoning and mathematical logic. We introduce the essential terminology below and refer to [16] for a more detailed presentation.

A many sorted signature Σ is a pair (S,F), where S is a set of *sorts* and F is a finite set of function symbols. For short, a many sorted signature Σ will be simply denoted by F. We assume that we have a partition of F in two subsets, the first one, C, contains the *constructor symbols* and the second, D, is the set of *defined symbols*. The symbol $\equiv$ is used for syntactic equality between two objects. The symbol $\setminus$ is used for set difference, i.e., $a \in (S \setminus T)$ if and only if $a \in S$ and $a \notin T$.

Let X be a family of sorted variables and let $T(F,X)$ be the set of well-sorted F-terms. $var(t)$ stands for the set of all variables appearing in t and $\sharp(x,t)$ denotes the number of occurrences of the variable x in t. A variable x in t is *linear* iff $\sharp(x,t) = 1$. A term t is linear if all its variables are linear. If $var(t)$ is empty then t is a *ground* term. By $T(F)$ we denote the set of all ground terms. From now on, we assume that there exists at least one ground term of each non-parameter sort.

Let N^* be the set of sequences of positive integers. For any term t, $occ(t) \subseteq N^*$ denotes its set of positions and the expression t/u denotes the *subterm of t at a position u*. We write $t[s]_u$ (resp. $t[s]$) to indicate that s is a subterm of t at position u (resp. at some position). The top position is written ε. Let $t(u)$ denote the symbol of t at position u. A position u in a term t is said to be *a strict position* if $t(u) \equiv f \in F$, a *linear variable position* if $t(u) \equiv x \in X$ and $\sharp(x,t) = 1$, a *non-linear variable position* if $t(u) \equiv x \in X$ and $\sharp(x,t) > 1$. The set of variable positions of a term t will be denoted by *var_pos(t)*.

If u is a position, then $|u|$ (the *length* of the corresponding string) gives us its *depth*. If t is a term, then the *depth* of t is the maximum of the depths of the positions in t and denoted *depth*(t) or abusively $|t|$. The *strict depth* of t, written as *sdepth*(t), is the maximum of the depths of the strict positions in t.

A F-substitution assigns F-terms of appropriate sorts to variables. Composition of substitutions σ and η is written by $\sigma\eta$. The F-term $t\eta$ obtained by applying a substitution η to t is called an *instance* of t. If η applies every variable of its domain to a ground term then we say that η is a ground substitution. If $t\eta$ is ground then it is a *ground instance* of t. A term t unifies with a term s if there exists a substitution σ such that $t\sigma \equiv s\sigma$.

A *conditional* F-equation is a F-equation of the following form: $s_1 = t_1 \wedge \cdots \wedge s_n = t_n \Rightarrow s_0 = t_0$, where $n \geqslant 0$ and $s_i, t_i \in T(F,X)$ are terms of the same sort. A F-clause is an expression of the form: $\neg(s_1 = t_1) \vee \neg(s_2 = t_2) \vee \cdots \vee \neg(s_n = t_n) \vee (s'_1 = t'_1) \vee \cdots \vee (s'_m = t'_m)$. When F is clear from the context we omit the prefix F. In the following, we consider a clause as a term in an extended alphabet. A clause is *positive* if $\neg$ does not occur in it. Let c_1 and c_2 be two clauses such that $c_1\sigma$ is a subclause of c_2 for some substitution σ, then we say that c_1 subsumes c_2. Let H be a set of clauses and c be a clause, we say that c is a logical consequence of H if c is valid in any model of H. This will be denoted by $H \models c$.

In the following, we suppose that $\prec$ is a transitive irreflexive relation on the set of terms that is noetherian, monotonic ($s \prec t$ implies $w[s] \prec w[t]$), stable ($s \prec t$ implies $s\sigma \prec t\sigma$) and satisfies the subterm property ($t \prec f(\ldots,t,\ldots)$, for all $t \in T(F,X)$). We also assume that the ordering $\prec$ can be extended consistently when adding new constants to the signature. The multiset extension of $\prec$ will be denoted by $\ll$. Let $\prec_c$ be a well-founded ordering on clauses that is monotonic, stable and satisfies the subterm property (see for instance [7]).

A conditional equation $a_1 = b_1 \wedge \cdots \wedge a_n = b_n \Rightarrow l = r$ will be written as $a_1 = b_1 \wedge \cdots \wedge a_n = b_n \Rightarrow l \to r$ if $\{l\sigma\} \gg \{r\sigma, a_1\sigma, b_1\sigma, \ldots, a_n\sigma, b_n\sigma\}$ for each substitution σ and $var(l)$ contains $var(r) \cup var(p)$, where $p \equiv \bigwedge_{i=1}^{n} a_i = b_i$; in that case we say that $a_1 = b_1 \wedge \cdots \wedge a_n = b_n \Rightarrow l \to r$ is a *conditional rule*. The term l is the *left-hand side* of the rule. A set of conditional rules is a *rewrite system*. Let R be a rewrite system and let c be a constructor symbol. If for any rule $r \in R$, the top of the left-hand side of r is different to c, then we say that c is a *free constructor*. A rewrite rule $p \Rightarrow l \to r$ is *left-linear* if l is linear. A rewrite system R is *left-linear* if every rule in R is left-linear, otherwise R is said to be *non-left-linear*. The *depth of a rewrite system* R, denoted *depth*(R), is defined as the maximum of the depths of the left-hand sides of R. Similarly, the *strict depth* of R denoted by *sdepth*(R), is the maximum of the depths of the strict positions in the left-hand sides of R.

From now on, we assume that for each conditional rule $p \Rightarrow l \to r$, if $l \in T(C,X)$, then $r \in T(C,X)$. A conditional rule is used to rewrite terms by replacing an instance of the left-hand side with the corresponding instance of the right-hand side (but not in the opposite direction) provided the conditions hold. The conditions are checked recursively. Termination is ensured because the conditions are smaller (w.r.t. $\prec$) than the conclusion. A set of conditional rules is called a conditional rewrite system. We can define the one-step rewrite relation $\to_R$ as follows:

Definition 2.1 (*Conditional rewriting*). Let R be a set of conditional equations. Let t be a term and u a position in t. We write: $t[l\sigma]_u \to_R t[r\sigma]_u$ if there is a substitution σ and a conditional equation $\bigwedge_{i=1}^{n} a_i = b_i \Rightarrow l = r$ in R such that:

1. $r\sigma \prec l\sigma$,
2. for all $i \in [1 \ldots n]$ there exists c_i such that $a_i\sigma \to_R^* c_i$ and $b_i\sigma \to_R^* c_i$,
3. $\{t[l\sigma]_u\} \gg \{a_1\sigma, b_1\sigma, \ldots, a_n\sigma, b_n\sigma\}$,

where $\to_R^*$ denotes the reflexive and transitive closure of $\to_R$.

A term t is *R-irreducible* (or in *normal form*) if there is no term s such that $t \rightarrow_R s$. Let R be a set of conditional rules. A term t is *strongly* irreducible by R (or *strongly R-irreducible*) if none of its non-variable subterms matches a left-hand side of R. Otherwise, we say that t is *strongly reducible* by R. We say that two terms s and t are joinable, if $s \rightarrow_R^* v$ and $t \rightarrow_R^* v$ for some term v. The rewrite relation $\rightarrow_R$ is said to be noetherian if there is no infinite chain of terms $t_1, t_2, \ldots, t_k, \ldots$ such that $t_i \rightarrow_R t_{i+1}$ for all i. The rewrite relation $\rightarrow_R$ is said to be *ground convergent* if the terms u and v are joinable whenever $u, v \in T(F)$ and $R \models u = v$.

3. Parameterized conditional specifications

A parameterized conditional specification is a pair $PS = (PAR, BODY)$ of specifications: $PAR = (F_{PAR}, E_{PAR})$ and $BODY = (F_{BODY}, E_{BODY})$, where F_{PAR} and F_{BODY} are signatures, E_{PAR} is the set of parameter *constraints* consisting of equational clauses over F_{PAR}, and E_{BODY} is the set of axioms of the parameterized specification. We assume that these axioms are conditional rules over $F = F_{PAR} \cup F_{BODY}$. Moreover, we assume that we have a partition of F_{BODY} in two subsets, the first one, C_{BODY}, contains the *constructor symbols* and the second, D_{BODY}, is the set of *defined symbols.*

Example 3.1. We consider the parameter specification *ordered elements* (see Fig. 1). In this example we have

$S_{PAR} = \{elem,\ bool\}$

$F_{PAR} = \{True,\ False,\ \leqslant,\ dif\}$

E_{PAR} is the set of *constraints* given in Fig. 1.

Let us consider the parameterized specification *ordered lists* with the formal parameter *ordered elements* (see Fig. 2). Here we have

$S = S_{PAR} \cup S_{BODY}$

$F = F_{PAR} \cup C_{BODY} \cup D_{BODY}$

where:

$S_{BODY} = \{nat,\ list\}$

$C_{BODY} = \{0,\ s,\ nil,\ Cons\}$

$D_{BODY} = \{length,\ count,\ insert,\ isort,\ sorted\}$

E_{BODY} is the set of *conditional rules* given in Fig. 2.

To prove the termination of E_{BODY}, we can use the *lexicographic path ordering* $\prec$ (see for instance [15]) with the following precedence on functions:

$$False \prec True \prec 0 \prec s \prec Nil \prec Cons \prec dif \prec count \prec length \prec \leqslant \prec sorted \prec insert \prec isort$$

parameter specification: ordered_elements

sorts elem, bool;

functions:
True : → bool;
False : → bool;
_≤_ : elem elem → bool;
*dif*__ : elem elem → bool;

constraints:
True = False ⇒;
x ≤ x = True;
x ≤ y = True ∨ x ≤ y = False;
x ≤ y = True ∨ y ≤ x = True;
x ≤ y = False ∨ y ≤ z = False ∨ x ≤ z = True;
dif(x,x)=False;
dif(x,y)=dif(y,x);
dif(x,y)=True ∨ dif(x,y)=False;
dif(x,y)=True ∨ dif(y,z)=True ∨ dif(x,z)=False;
dif(x,y)=True ∨ y ≤ z = False ∨ x ≤ z = True;
x ≤ y = False ∨ dif(y,z) = True ∨ x ≤ z = True;
dif(x,y)=False ∨ x ≤ y = False ∨ y ≤ z = False ∨ dif(x,z)=True;
x ≤ y = False ∨ y ≤ z = False ∨ dif(y,z)=False ∨ dif(x,z)=True;
end

Fig. 1. The parameter specification ordered elements.

3.1. The canonical term algebra

An actualization [17] of the parameter theory E_{PAR} is a model $\mathscr{A}$ of E_{PAR}. In order to be able to integrate an actualization $\mathscr{A}$ of the parameter theory into the rewrite process, we describe $\mathscr{A}$ by its so-called *diagram* [17]. For this reason we enrich the signatures by adding new constants $\underline{a}$ for each element a of the carrier A of $\mathscr{A}$. Let $\mathscr{N}(A)$ be the set of new constants and let $F(\mathscr{A}) = F \cup \mathscr{N}(\mathscr{A})$. The diagram $\mathscr{D}(\mathscr{A})$ of $\mathscr{A}$ is the set of (directed) equations $f(\underline{a}_1, \ldots, \underline{a}_n) = \underline{a}$ such that $f \in F_{PAR}$; a_i, $a \in A$ and $f^{\mathscr{A}}(a_1, \ldots, a_n) = a$. We denote by $E_{BODY}(\mathscr{A})$ the set $E_{BODY} \cup \mathscr{D}(\mathscr{A})$. For any model $\mathscr{A}$ of E_{PAR}, we define a canonical term algebra $\mathscr{T}(\mathscr{A})$ representing the semantics of the result of an actualization: $\mathscr{T}(\mathscr{A}) = T(F(\mathscr{A}))/_{=_{E_{BODY}(\mathscr{A})}}$, where $=_{E_{BODY}(\mathscr{A})}$ is the smallest congruence on $T(F(\mathscr{A}))$ generated by $E_{BODY(\mathscr{A})}$. An interesting case is when $\mathscr{T}(\mathscr{A})$ is an initial model in the class of $F(\mathscr{A})$-algebras that are models of $E_{BODY}(\mathscr{A})$ for any model $\mathscr{A}$ of E_{PAR}. To guarantee this fact we need that $E_{BODY}(\mathscr{A})$ is consistent (i.e. has a model) for any model $\mathscr{A}$ of E_{PAR}. This result is shown by the following theorem which is analogous to Theorem 2.8 from [36].

Theorem 3.2. *If $E_{BODY}(\mathscr{A})$ is consistent for any model $\mathscr{A}$ of E_{PAR}, then $\mathscr{T}(\mathscr{A})$ is initial in the class of $F(\mathscr{A})$-algebras that are models of $E_{BODY}(\mathscr{A})$ for any model $\mathscr{A}$ of E_{PAR}.*

```
specification: ordered_lists

parameters: ordered_elements;

sorts nat, list;

constructors:
0          :            → nat;
s_         : nat        → nat;
Nil        :            → list;
Cons_ _    : elem list  → list;

defined functions:
length_    : list       → nat;
count_ _   : elem list  → nat;
insert_ _  : elem list  → list;
isort_     : list       → list;
sorted_    : list       → bool;

axioms:
length(Nil) → 0;
length(Cons(x,l)) → s(length(l));
count(x,Nil) → 0;
dif(x,y)=False ⇒ count(x,Cons(y,z)) → s(count(x,z));
dif(x,y)=True ⇒ count(x,Cons(y,z)) → count(x,z);
insert(x,Nil) → Cons(x,Nil);
x ≤ y=True ⇒ insert(x,Cons(y,z)) → Cons(x,Cons(y,z));
x ≤ y=False ⇒ insert(x,Cons(y,z)) → Cons(y,insert(x,z));
isort(Nil) → Nil;
isort(Cons(x,l)) → insert(x,isort(l));
sorted(Nil) → True;
sorted(Cons(x,Nil)) → True;
x ≤ y=False ⇒ sorted(Cons(x,Cons(y,z))) → False;
x ≤ y=True ⇒ sorted(Cons(x,Cons(y,z))) → sorted(Cons(y,z));
```

Fig. 2. The parameterized specification ordered lists.

The problem of checking consistency of parameterized specifications is not addressed in this paper. However, much work has been concerned with checking this property (see for instance [17, 36, 27, 3]).

3.2. *Proving inductive theorems w.r.t. parameterized specifications*

We shall now define what is an inductive theorem in parameterized specifications. Note that the theorems to be proved are *clauses*.

Definition 3.3. Let PS be a parameterized specification. A *clause* Γ is an inductive theorem of PS (or inductively valid), iff $\mathcal{T}(\mathcal{A})$ is a model of Γ for any model $\mathcal{A}$ of E_{PAR}. This will be denoted by $PS \models_{ind} \Gamma$ or $E_{BODY}(\mathcal{A}) \models_{ind} \Gamma$ for any model $\mathcal{A}$ of E_{PAR}.

The next lemma which is similar to Lemma 9 from [3], gives us a useful characterization of inductive theorems.

Lemma 3.4. *A clause* $\Gamma \equiv \neg(u_1 = v_1) \vee \cdots \vee \neg(u_n = v_n) \vee (s_1 = t_1) \vee \cdots \vee (s_m = t_m)$ *is an inductive theorem of PS iff for any model* $\mathscr{A}$ *of* E_{PAR} *and for any ground substitution* σ *over* $T(F(\mathscr{A}))$:

$$(\forall i\colon E_{BODY}(\mathscr{A}) \models u_i\sigma = v_i\sigma) \text{ implies } (\exists j\colon E_{BODY}(\mathscr{A}) \models s_j\sigma = t_j\sigma)$$

4. Sufficient completeness for parameterized specifications

The property of sufficient completeness is in general undecidable. We now give a method for testing this property for conditional parameterized specifications. This method is inspired by [28, 9] and is based on the notion of *Pattern* trees. Let $\mathscr{A}$ be a model of E_{PAR}. If any ground term in $T(F(\mathscr{A}))$ can be expressed with only constructors and elements of $\mathscr{N}(\mathscr{A})$, we say that PS is complete w.r.t. the constructors and parameter (or sufficiently complete). Here is a more formal definition:

Definition 4.1 (*Sufficient completeness*). We say that $PS = (PAR, BODY)$ is sufficiently complete if and only if for any model $\mathscr{A}$ of E_{PAR}, for all t in $T(F(\mathscr{A}))$ there exists t' in $T(C_{BODY} \cup \mathscr{N}(\mathscr{A}))$ such that $t \to^*_{E_{BODY}(\mathscr{A})} t'$.

4.1. How to check sufficient completeness

The main idea behind our test for sufficient completeness is to compute a *pattern tree* for every f in D_{BODY}. A pattern tree for f is a tree whose nodes are terms, whose root is $f(x_1,\ldots,x_n)$, where n is the arity of f and $x_1,\ldots,x_n$ are distinct variables, and such that the successors of any internal node $t \equiv f(t_1,\ldots,t_n)$ are obtained by replacing a variable which appears at an *extension position* of t by all possible terms $c(y_1,\ldots,y_m)$, where c is a constructor symbol and $y_1,\ldots,y_m$ are new distinct variables not already in t. The restriction of extension positions permits us to build a pattern tree which captures the structure of the parameterized specification. The leaves of the tree give a case analysis on the arguments of f. If all leaves are "pseudo-reducible by PS", therefore, by Theorem 4.11, the answer is affirmative. To compute pattern trees, we use the notions of *nullary sort* and *nullary variable position* which can be defined as follows.

Definition 4.2 (*Nullary sort, nullary variable position*). A sort $\mathscr{S}$ is *nullary*,[3] if for any model $\mathscr{A}$ of E_{PAR}, there is no infinite set of terms in $T(F(\mathscr{A}))$ of sort $\mathscr{S}$ that are irreducible by $E_{BODY}(\mathscr{A})$. Let t be a term and u a variable position in t, u is *nullary* if the sort of $t(u)$ is *nullary*.

[3] This property is decidable if the functions in D_{BODY} are sufficiently complete over free constructors.

The following definition characterizes induction positions of function symbols in F_{BODY}.

Definition 4.3 (*Induction positions*). For $f \in F_{BODY}$ we define the set of induction positions of f as follows: $ind\_pos(f) = \{u \mid$ there is $p \Rightarrow g \rightarrow d \in E_{BODY}$ such that $g(\varepsilon) \equiv f$ and u is either a strict and non-top position in g or a non-parameter and non-linear variable[4] position in $g\}$.

Example 4.4. Consider the following parameterized specification:

$$S_{PAR} = \{elem\}$$

$$F_{PAR} = \emptyset$$

$$E_{PAR} = \emptyset$$

$$S_{BODY} = \{nat,\ card\}$$

$$C_{BODY} = \{0 :\rightarrow nat,\ s : nat \rightarrow nat,\ R :\rightarrow card,\ B :\rightarrow card\}$$

$$D_{BODY} = \{f : nat \times nat \times nat \rightarrow nat,\ g : card \times card \rightarrow nat,$$

$$h : elem \times elem \rightarrow nat\}$$

E_{BODY} contains the following conditional rules:

$$f(x, y, x) \rightarrow x$$

$$f(x, x, y) \rightarrow x$$

$$f(y, x, x) \rightarrow x$$

$$g(z, t) \rightarrow 0$$

$$h(e, e) \rightarrow 0$$

where x and y are two variables of sort *nat*, z and t are two variables of sort *card* and e is a variable of sort *elem*.

- $ind\_pos(f) = \{1, 2, 3\}$, the positions 1, 2 and 3 are induction positions since they are non-parameter and non-linear variable positions.
- $ind\_pos(g) = \emptyset$, the positions 1 and 2 are not induction positions since they are neither strict positions nor non-parameter and non-linear variable positions.
- $ind\_pos(h) = \emptyset$, the positions 1 and 2 are not induction positions since they are parameter variable positions.

Example 4.5 (*Example 3.1 continued*). The output of the SPIKE procedure that computes induction positions of functions is given in Fig. 3.

[4] A non-parameter variable is a variable of non-parameter sort.

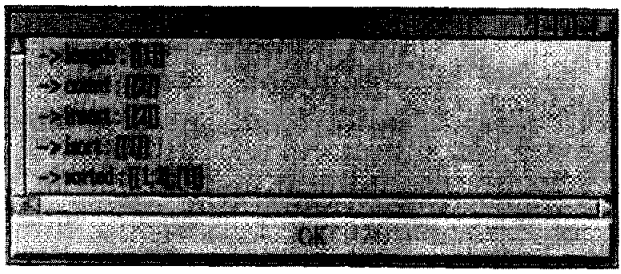

Fig. 3. Induction positions of functions

From any node of the tree labelled by the term $t \equiv f(t_1, \ldots, t_n)$, with $t_i \in T(C_{BODY}, X)$ for all $i \in [1 \ldots n]$, we build the sons of this node by choosing a variable position u of t that is *nullary* or that is an *induction position of* f and by making a graft at this position. Each son is thereby labelled by an element of a set of terms called $sons(t, u)$. In this case, we say that t is extensible.

Definition 4.6. Let t be a term of the form $f(t_1, \ldots, t_n)$ where, for all i, $t_i \in T(C_{BODY}, X)$. Let u be a variable position of t, that is nullary or that belongs to $ind\_pos(f)$. Suppose that $t(u)$ is of sort s. We define $sons(t, u)$ as follows: $sons(t, u) = \{t[c]_u \mid c \equiv c_i(x_1, \ldots, x_n)$ where c_i is a constructor with codomain s, and arity n, and $x_1, \ldots, x_n$ are new distinct variables not already in $t\}$.

We say that u is an extension position and that t is extensible. The transformation operation of t to $sons(t, u)$ is called the graft of t at the position u. We denote by $ext\_pos(t)$ the set of extension positions of t.

Example 4.7 (*Example 4.4 continued*). Let $t \equiv f(x, y, z)$ and $t' \equiv g(x, y)$ then

$$sons(t, 2) = \{f(x, 0, z),\ f(x, s(y), z)\} \text{ and } sons(t', 2) = \{g(x, R),\ g(x, B)\}$$

Note that 2 is an *extension position* in t' since the sort *card* is *nullary*: the only constructors of sort *card* are R and B.

The construction of the pattern tree is based on the notion of *case rewriting* which can be defined as follows.

Definition 4.8 (*Case rewriting*). Let PS be a parameterized specification and let t be a term. Assume there exists a non-empty sequence of conditional rules $p_1 \Rightarrow t_1 \rightarrow r_1$, $p_2 \Rightarrow t_2 \rightarrow r_2, \ldots, p_n \Rightarrow t_n \rightarrow r_n$ in E_{BODY} and a sequence of positions $u_1, u_2, \ldots, u_n$ in t such that $t/u_1 \equiv t_1\sigma_1$, $t/u_2 \equiv t_2\sigma_2, \ldots, t/u_n \equiv t_n\sigma_n$ and $p_1\sigma_1 \vee p_2\sigma_2 \vee \cdots \vee p_n\sigma_n$ is an inductive theorem of PS. Then, we write

$$case\_rewriting(t) = \{p_1\sigma_1 \Rightarrow t[r_1\sigma_1]_{u_1}, \ldots, p_n\sigma_n \Rightarrow t[r_n\sigma_n]_{u_n}\}$$

In this case, t is said to be *pseudo-reducible* by PS. Otherwise, t is said to be *pseudo-irreducible* by PS.

Thus, if a term t is *pseudo-reducible* by PS, then all its ground instances are reducible.

Example 4.9 (*Example 3.1 continued*). Consider the function *smaller*:

$$smaller : elem \times list \rightarrow bool$$

The expression $smaller(x, l)$ means that x is smaller than every element of the list l. We can define *smaller* by the following axioms:

$$smaller(x, Nil) \rightarrow True$$

$$x \leqslant y = True \wedge smaller(x, l) = True \Rightarrow smaller(x, Cons(y, l)) \rightarrow True$$

$$smaller(x, l) = False \Rightarrow smaller(x, Cons(y, l)) \rightarrow False$$

$$x \leqslant y = False \Rightarrow smaller(x, Cons(y, l)) \rightarrow False$$

The term $smaller(x, Cons(y, l))$ is *pseudo-reducible* by PS since

$$(x \leqslant y = True \wedge smaller(x, l) = True) \vee (smaller(x, l) = False) \vee (x \leqslant y = False)$$

is an inductive theorem of PS. However, the term $smaller(x, l)$ is *pseudo-irreducible* by PS.

It is useless to continue the graft process when we meet a node labelled by a term which is pseudo-reducible by PS. Thus, we can describe in the following way the construction of the pattern tree from the tree initially constituted from the root $t \equiv f(x_1, \ldots, x_n)$, where n is the arity of f and $x_1, \ldots, x_n$ are distinct variables. We first check the pseudo-reducibility by PS of t. If t is pseudo-irreducible by PS, then we build at every step the sons of a node s of the tree by choosing a position in $ext\_pos(s)$ and by making a graft operation on s at this position. The construction of the tree stops if each of its sons is either pseudo-reducible by PS or we can split it no further.

4.2. Construction rules

To check if an operator f in D_{BODY} is sufficiently complete, we apply the rules given in Fig. 4. *Candidates* is the set of terms used for the reducibility check. *Red* is the set of leaves of the tree which are pseudo-reducible. *Irred* is the set of leaves of the tree which are pseudo-irreducible and not extensible. The initial state is $(\{f(x_1, \ldots, x_n)\}, \emptyset, \emptyset)$, where n is the arity of f and $x_1, \ldots, x_n$ are distinct variables. The rule *stop* is applied

stop:

$(\emptyset, Red, Irred) \vdash_C stop$

delete reducible leaf:

$(Candidates \cup \{t\}, Red, Irred) \vdash_C (Candidates, Red \cup \{t\}, Irred)$

if t is pseudo-reducible

decompose:

$(Candidates \cup \{t\}, Red, Irred) \vdash_C (Candidates \cup sons(t,u), Red, Irred)$

if t is pseudo-irreducible and $u \in ext\_pos(t)$.

delete irreducible leaf:

$(Candidates \cup \{t\}, Red, Irred) \vdash_C (Candidates, Red, Irred \cup \{t\})$

if t is pseudo-irreducible and $ext\_pos(t) = \emptyset$.

Fig. 4. Transformation System. C.

if the set *candidates* is empty. Then, if *Irred* is empty, we conclude that all the leaves of the pattern tree are pseudo-reducible by *PS*. Consequently, the operator f is sufficiently complete (see Theorem 4.11). If we meet a term t that is pseudo-reducible by *PS*, then the *delete-reducible-leaf* rule adds it to the set *Red* and we continue the check of the pseudo-reducibility of the other leaves of the tree. The *decompose* rule expresses the operation of decomposition of a term t at the position u. This rule applies if we meet a term t that is extensible and pseudo-irreducible by *PS*. The graft operation produces the sons of t, for which we must check pseudo-reducibility. Finally, the *delete-irreducible-leaf* rule is applied if we meet a leaf of the tree that is not extensible and pseudo-irreducible by *PS*. In this case we add the term t to the set *Irred* and we continue the check of the pseudo-reducibility of the other leaves of the tree.

The height of the pattern tree is bounded. This result is shown by the following lemma.

Lemma 4.10. *Let t be a term $f(x_1,\ldots,x_n)$, where $f \in D_{BODY}$ and $x_1,\ldots,x_n$ are distinct variables. The pattern tree of f, computed by C, is bounded.*

Proof. The rules of E_{BODY} which have the function symbol f at the top is finite. This means that the set $ind\_pos(f)$ is finite too. As a consequence the set $var\_pos(t) \cap ind\_pos(f)$ decreases during the construction of the tree since consecutive grafts in the same branch of the tree are made at deeper and deeper positions. On the other hand, a nullary position corresponds to a finite set of constructor terms. Consequently, the height of the pattern tree is bounded.

4.2.1. Soundness and completeness

In the following we denote by $C_{BODY}(\mathscr{A})$ the set $C_{BODY} \cup \mathscr{N}(\mathscr{A})$.

Theorem 4.11. *Given a parameterized specification $PS = (PAR, BODY)$ such that $\rightarrow_{E_{BODY}(\mathscr{A})}$ is ground convergent*[5] *for every model $\mathscr{A}$ of E_{PAR}, if for all f in D_{BODY}, there exists a sequence of states $(\{f(x_1,\ldots,x_n)\}, \emptyset, \emptyset) \vdash_C \cdots \vdash_C (\emptyset, Red, \emptyset)$, then PS is sufficiently complete.*

Proof. Let f be a function symbol and suppose that there exists a sequence of states

$$(\{f(x_1,\ldots,x_n)\},\emptyset,\emptyset) \vdash_C \cdots \vdash_C (\emptyset, Red, \emptyset)$$

Let $\mathscr{A}$ be a model of E_{PAR}; we have to prove the following property:

$$\mathscr{P}(t)\text{: } \forall t \in T(F(\mathscr{A})), \quad \exists t' \in T(C_{BODY}(\mathscr{A})) \text{ such that } t \rightarrow^*_{E_{BODY}(\mathscr{A})} t'$$

We proceed by induction on t w.r.t. $\prec$ which contains $\rightarrow_{E_{BODY}(\mathscr{A})}$. Without loss of generality, we can assume that $t \equiv f(t_1,\ldots,t_n)$ where $f \in D_{BODY}$ and, for all i, t_i is in $C_{BODY}(\mathscr{A})$. Then, there exists a leaf s of the pattern tree and a ground substitution σ over $T(C_{BODY}(\mathscr{A}))$ such that $s\sigma \equiv t$. Since s must be pseudo-reducible by PS, there exists a non-empty sequence of conditional rules $p_1 \Rightarrow t_1 \rightarrow r_1$, $p_2 \Rightarrow t_2 \rightarrow r_2, \ldots, p_n \Rightarrow t_n \rightarrow r_n$ in E_{BODY} and a sequence of positions $u_1, u_2, \ldots, u_n$ in s such that $s/u_1 \equiv t_1\sigma_1$, $s/u_2 \equiv t_2\sigma_2, \ldots, s/u_n \equiv t_n\sigma_n$ and $p_1\sigma_1 \vee p_2\sigma_2 \vee \cdots \vee p_n\sigma_n$ is an inductive theorem of PS. Then, there exists k such that $PS \models_{ind} p_k\sigma_k$ and therefore $t_k\sigma_k \rightarrow_{E_{BODY}(\mathscr{A})} r_k\sigma_k$ since, for every model $\mathscr{A}$ of E_{PAR}, $\rightarrow_{E_{BODY}(\mathscr{A})}$ is ground convergent over $T(F(\mathscr{A}))$. On the other hand; $\rightarrow_{E_{BODY}(\mathscr{A})}$ is stable by substitution;[6] therefore $t_k\sigma_k\sigma \rightarrow_{E_{BODY}(\mathscr{A})} r_k\sigma_k\sigma$. Finally, we have $r_k\sigma_k\sigma \prec t$ and $r_k\sigma_k\sigma \in T(F(\mathscr{A}))$ since $p_k \Rightarrow t_k \rightarrow r_k$ is a conditional rule. Then by the induction hypothesis, we conclude that there exists t' in $T(C_{BODY}(\mathscr{A}))$ such that $r_k\sigma_k\sigma \rightarrow^*_{E_{BODY}(\mathscr{A})} t'$ and therefore there exists t'' in $T(C_{BODY}(\mathscr{A}))$ such that $t \rightarrow^*_{E_{BODY}(\mathscr{A})} t''$.

In the following completeness proof of the procedure, we assume without loss of generality that all parameter variables on the left-hand sides of E_{BODY} are linear. In fact, we can easily transform a rule which contains non-linear parameter variables on its left-hand side to an equivalent conditional left-linear rule w.r.t. parameter variables. For instance, in Example 3.1, the rule: $count(x, Cons(x,z)) \rightarrow s(count(x,z))$ is equivalent to: $dif(x,y) = False \Rightarrow count(x, Cons(y,z)) \rightarrow s(count(x,z))$.

Theorem 4.12. *Let $PS = (PAR, BODY)$ be a parameterized specification. Suppose that the constructors are free and if the defined function g appears on a left-hand*

[5] Some works have been concerned with checking the ground convergence of a rewrite relation associated with a parameterized specification (see for instance [3]).

[6] For all substitutions ϕ: $t \rightarrow_{E_{BODY}(\mathscr{A})} t'$ implies $t\phi \rightarrow_{E_{BODY}(\mathscr{A})} t'\phi$.

side of a conditional rule in E_{BODY}, then every rule in E_{BODY} that contains g on its left-hand side is linear. If PS is sufficiently complete, then for all f in D_{BODY}, there exists a sequence of states $(\{f(x_1,\ldots,x_n)\},\emptyset,\emptyset) \vdash_C \cdots \vdash_C (\emptyset, Red, \emptyset)$.

Proof. Assume that *PS* is sufficiently complete. Suppose that there exists a leaf t which is not extensible and pseudo-irreducible by *PS*. Then, there are two cases to be considered:

(a) t is *strongly irreducible* by E_{BODY}. Let $\mathscr{A}$ be a model of E_{PAR} and assume that $\{x_1,\ldots,x_k\}$ is the set of non-parameter variables of t. Let us consider a ground substitution ϕ such that, for all parameter variables x of t, $x\phi$ is an element from $\mathscr{N}(\mathscr{A})$ of the same sort as x, and, for all $i \in [1\ldots k]$, $x_i\phi$ is strongly $E_{BODY}(\mathscr{A})$-irreducible, and:

1. $\forall i \in [1\ldots k], |x_i\phi| > |t|$,
2. $\forall i,j \in [1\ldots k], i \neq j, \quad ||x_i\phi| - |x_j\phi|| > |t|$.

Note that such ϕ exists thanks to the fact that t is not extensible and the constructors are free, so we can choose $x_i\phi$ among the terms built from constructor symbols and elements of $\mathscr{N}(\mathscr{A})$. Assume now that $t\phi$ contains an instance of a left-hand side g of a rule in E_{BODY}. Since any $x_i\phi$ is strongly E_{BODY}-irreducible, there is a strict position u in t such that $t\phi/u$ is an instance of g. Let v be a position of g such that $g(v)$ is a function symbol. $t(uv)$ is a function symbol since t is not extensible. We consider two cases:

(a.1) Assume that g is linear. We can define a substitution σ such that for every variable x that occurs at position w of g we have $x\sigma \equiv t/uw$. Such a substitution exists by the linearity of g. We then have $t/u \equiv g\sigma$ which contradicts the assumption that t is strongly E_{BODY}-irreducible.

(a.2) Assume that g is non-linear. Since t is not an instance of g and $t(uw) \equiv g(w)$ for every strict position w of g. On the other hand, all parameter variables on the left-hand sides of E_{BODY} are linear. Therefore, there exists two positions u_1 and u_2 of a non-parameter variable x in g such that: $t/uu_1 \not\equiv t/uu_2$ and $t\phi/uu_1 \equiv t\phi/uu_2$. There are three cases to be considered:

(a.2.1) t/uu_1 and t/uu_2 are ground. In this case $t/uu_1 \equiv t\phi/uu_1$ and $t/uu_2 \equiv t\phi/uu_2$. Therefore, $t/uu_1 \equiv t/uu_2$, which is a contradiction.

(a.2.2) t/uu_1 is ground and t/uu_2 non-ground. Then some x_i occurs in t/uu_2. We have $|x_i\phi| > |t|$ by construction of ϕ and therefore $|t\phi/uu_2| > |t|$. On the other hand, $|t\phi/uu_2| = |t\phi/uu_1| = |t/uu_1| \leqslant |t|$, which is a contradiction.

(a.2.3) t/uu_1 and t/uu_2 are non-ground. Then there is a position v and a variable x_k such that $t/uu_1v \equiv x_k$ and $t/uu_2v \not\equiv x_k$.

- If t/uu_2v is ground the proof is similar to (a.2.2).
- If t/uu_2v is non-ground let $var(t/uu_2v) = \{y_1,\ldots,y_p\}$.
  - If $x_k \in var(t/uu_2v)$ then $|t\phi/uu_1v| < |t\phi/uu_2v|$ and therefore we cannot have $t\phi/uu_1 \equiv t\phi/uu_2$ as this leads to a contradiction.
  - If $x_k \notin var(t/uu_2v)$ then let y_j be the variable in $var(t/uu_2v)$ such that $|y_j\phi| = max_{l=1,\ldots,p}|y_l\phi|$.

(1) If $|x_k\phi| > |y_j\phi| + |t|$ then $|t\phi/uu_1v| = |x_k\phi| > |y_j\phi| + |t| > |t\phi/uu_2v|$.
(2) If $|y_j\phi| > |x_k\phi| + |t|$ then $|t\phi/uu_2v| \geqslant |y_j\phi| > |x_k\phi| + |t| > |t\phi/uu_1v| = |x_k\phi|$ and we derive a contradiction too.

Therefore, $t\phi$ is strongly irreducible by E_{BODY}. On the other hand, t does not contain any parameter functions, so $t\phi$ is ground and irreducible by $E_{BODY}(\mathscr{A})$. This contradicts the assumption.

(b) Otherwise, t is *strongly reducible* by E_{BODY}. Let $L = \{p_1 \Rightarrow l_1 \rightarrow r_1, \ldots, p_n \Rightarrow l_n \rightarrow r_n\}$ be the non-empty set of all conditional rules in E_{BODY} such that there exists $u_1, \ldots, u_n$ with $t/u_1 \equiv l_1\sigma_1, \ldots, t/u_n \equiv l_n\sigma_n$. Since t is pseudo-irreducible by PS, $p \equiv p_1\sigma_1 \vee \cdots \vee p_n\sigma_n$ is not an inductive theorem of PS. Thus there exists a model $\mathscr{A}$ of E_{PAR} and a substitution τ over $T(F(\mathscr{A}))$ such that $E_{BODY}(\mathscr{A}) \not\models_{ind} p\tau$.

Then, $t\tau$ cannot be reducible at the top. Assume otherwise that there exists a rule $r \in (E_{BODY} \setminus L)$ with left-hand side g that applies to $t\tau$ and $t\tau \equiv g\sigma$. Note that every non-variable position of g is a non-variable position of t since t is not extensible. On the other hand, g is linear by hypotheses. So we can define a substitution ρ by $x\rho \equiv t/w$ for every variable x that occurs at some position w of g. We have then $t \equiv g\rho$, in contradiction with the assumption that L contains all the rules whose left-hand side matches t.

The term $t\tau$ cannot be reducible at another position since no proper subterm of $t\tau$ contains a defined symbol and since the constructors are free, which is a contradiction.

4.3. *Sufficient completeness with* SPIKE

SPIKE checks automatically if an operator f in a specification PS is sufficiently complete. The program displays pattern trees that show a case analysis on the arguments of the defined functions. If all the leaves are *pseudo-reducible* by PS, the answer is affirmative. If one of the leaves is not extensible and *pseudo-irreducible* by PS, then SPIKE suggests new rules for completing the specification. These rules are not entirely determined but rather possible schemes for them are proposed; every rule is of the form: (*condition, left-hand side*). Once the user has chosen the new rules, usually by simply giving their right-hand sides, SPIKE replays the test. Consider Example 3.1 and suppose that all the rules for defining *sorted* are removed except for the rule:

$$x \leqslant y = True \Rightarrow sorted(Cons(x, Cons(y,z))) = sorted(Cons(y,z))$$

Then, *sorted* is not sufficiently complete. Here we describe a session with SPIKE to give an idea about the interaction with the user in order to recover a sufficiently complete specification (see Figs. 5 and 6). We therefore add three rules and try again (see Fig. 7).

Note that this procedure includes an inductive theorem proving to check the pseudo-reducibility of the leaves of the tree. Therefore, the efficiency of our procedure depends on that of the inductive theorem prover. In the following section, we propose a new procedure to prove and disprove inductive properties automatically in

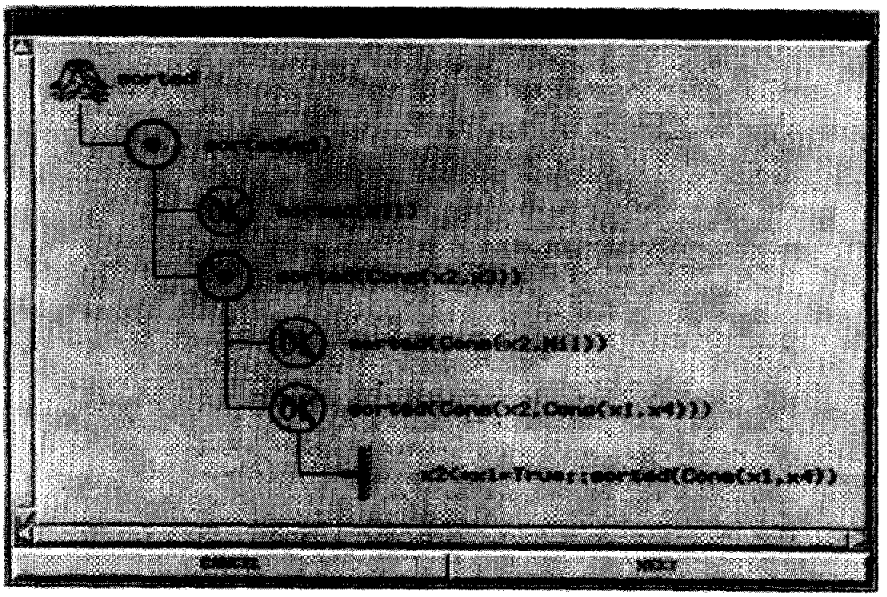

Fig. 5. The function *sorted* is not sufficiently complete.

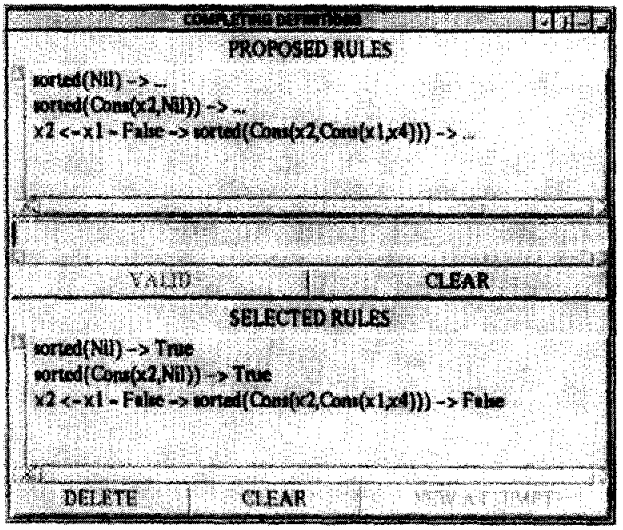

Fig. 6. Suggestions.

parameterized conditional specifications. It is worth emphasizing that proofs by induction in parameterized specifications allows for shorter and more structured proofs. Moreover, a generic proof must be given only once and can be reused for each instantiation of the parameter.

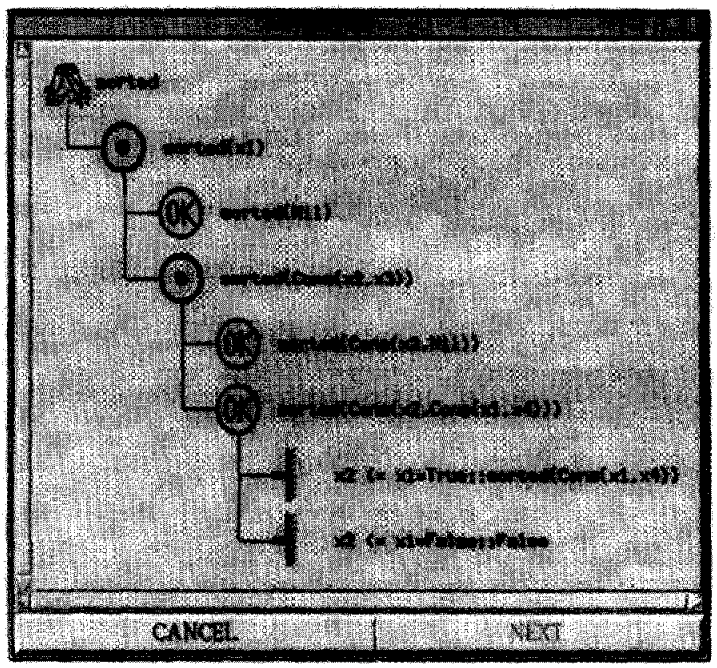

Fig. 7. The function *sorted* is now sufficiently complete.

5. Selection of induction schemes

To perform a proof by induction, it is necessary to provide some induction schemes. In our framework these schemes are defined *first* by a function which, given a conjecture, selects the positions of variables where induction will be applied and *second* by a special set of terms called a test set with which the induction variables are instantiated. In general, the selection of good induction positions leads to drastic improvements.

5.1. How to get induction variables

Given a specification, we start by computing a set of induction positions of function symbols (see Definition 4.3). This computation is done only once and it permits us to determine whether a variable position of a term t is an induction variable or not.

Definition 5.1 (*induction variable*). Let t be a term containing a variable x of non-parameter sort s. We say that x is an *induction variable* of t if s is *nullary*, if x occurs at a position $u.v$ of t such that v is an induction position of $t(u)$, or if x is t.

Example 5.2 (*Example 3.1 continued*). Let $t \equiv insert(x, insert\ (x, y))$, y is the only induction variable of t because y occurs at position 2 of the subterm $insert(x, y)$ and

position 2 is an induction position of *insert*. The term $insert(x, Cons(y, Cons(z, t)))$ does not contain any induction variables.

5.2. How to get test sets

A test set can be seen as a special induction scheme that permits us to refute false conjectures by the construction of a counter example. The definition of a test set given below is more general than the one in [8]. It permits us to refute false conjectures even if the constructors are not free. On the other hand, with the new definition, we obtain a smaller test set, which improves the efficiency of the proof procedure (see Section 7.1).

To define test set, we use the following notions: The *bound* for PS, denoted $D(PS)$, is equal to $depth(E_{BODY}) - 1$ if $sdepth(E_{BODY}) < depth(E_{BODY})$ and E_{BODY} is left linear, otherwise $D(PS)$ is equal to $depth(E_{BODY})$. We say that a term t is *infinitary* if for any model $\mathscr{A}$ of E_{PAR} and for any position u in t for which t/u is a non-ground term, there exists infinitely many strongly $E_{BODY}(\mathscr{A})$-irreducible ground instances of t whose subterms at position u are distinct.

Definition 5.3 (*Test set*). A test set $S(PS)$ for a parameterized specification PS is a finite set of terms over $T(F, X)$ that has the following properties:

1. For any model $\mathscr{A}$ of E_{PAR} and for any $E_{BODY}(\mathscr{A})$-irreducible term s in $T(F(\mathscr{A}))$, there exist a term t in $S(PS)$ and a ground substitution σ over $T(F(\mathscr{A}))$ such that $t\sigma \equiv s$;
2. any non-ground term in $S(PS)$ is *infinitary* and has non-parameter variables at depth greater than or equal to $D(PS)$.

The first property allows us to prove theorems by induction on the domain of irreducible terms rather than on the whole set of terms. Sets of terms with property 1. are usually called *cover sets* in the literature. Several proof procedures have been built on cover sets [39, 37]. Note that our method is also valid if we use cover sets rather than test sets. However, *cover sets* cannot be used to refute false conjectures. The second property of test sets is fundamental for this purpose (see Theorem 5.6). The next definition provides us with a criteria to reject false conjectures.

Definition 5.4 (*Quasi-inconsistent*). Given a parameterized specification $PS = (PAR, BODY)$ and a test set $S(PS)$, then a clause $C \equiv \neg(s_1 = t_1) \vee \cdots \vee \neg(s_m = t_m) \vee (g_1 = d_1) \vee \cdots \vee (g_n = d_n)$ is quasi-inconsistent with respect to PS if there is a test substitution[7] σ of C such that:

1. $E_{PAR} \not\models (g_1 = d_1 \vee \cdots \vee g_n = d_n)\sigma$;
2. for all $i \in [1 \ldots m]$: $s_i\sigma = t_i\sigma$ is an inductive theorem of PS;

[7] We say that a substitution σ is a test substitution of C if it maps every induction variable of C to an element of $S(PS)$ of the same sort and whose variables have been renamed.

3. for all $j \in [1 \ldots n]$: the maximal[8] elements of $\{g_j\sigma, d_j\sigma\}$ w.r.t. $\prec$ are strongly E_{BODY}-irreducible.

Example 5.5 (*Example 3.1 continued*). The conjecture:

$$C \equiv x \leqslant y = False, x \leqslant z = False, y \leqslant z = True$$

is quasi-inconsistent. In fact, C do not contain any induction variable and we have:
- $E_{PAR} \not\models C$
- $x \leqslant y$, $x \leqslant z$ and $y \leqslant z$ are strongly E_{BODY}-irreducible.

The next result gives us sufficient conditions under which a quasi-inconsistent clause cannot be inductively valid. This is proved by building a well-chosen ground instance of the clause which gives us a counterexample.

Theorem 5.6. *Given a parameterized specification* $PS = (PAR, BODY)$ *such that* $\rightarrow_{E_{BODY}(\mathscr{A})}$ *is ground convergent for every model* $\mathscr{A}$ *of* E_{PAR}, *if a clause* C *is quasi-inconsistent, then* C *is not an inductive theorem of* PS.

Proof. Let $C \equiv \neg(s_1 = t_1) \vee \cdots \vee \neg(s_m = t_m) \vee (g_1 = d_1) \vee \cdots \vee (g_n = d_n)$ be a clause which is quasi-inconsistent with respect to PS. Then there is a test substitution σ of C such that $E_{PAR} \not\models (g_1 = d_1 \vee \cdots \vee g_n = d_n)\sigma$ and for all $i \in [1 \ldots m]$, $PS \models_{ind} (s_i = t_i)\sigma$.

Let $\mathscr{M}$ be the set of the maximal elements of $\{g_j\sigma, d_j\sigma\}$ w.r.t. $\prec$. Then every element in $\mathscr{M}$ is strongly E_{BODY}-irreducible. In order to show that C is not an inductive theorem of PS, it is sufficient to show that there exists a model $\mathscr{A}$ of E_{PAR} and a ground substitution β over $T(F(\mathscr{A}))$ such that $E_{BODY}(\mathscr{A}) \not\models_{ind} (g_1 = d_1 \vee \cdots \vee g_n = d_n)\beta$ since every ground instance of $\neg(s_1 = t_1) \vee \cdots \vee \neg(s_m = t_m)$ is not inductively valid.

Let $Q \equiv (g_1 = d_1 \vee \cdots \vee g_n = d_n)$. We have $E_{PAR} \not\models Q\sigma$; thus there exists a model $\mathscr{A}$ of E_{PAR} and a substitution τ over $T(\mathscr{N}(\mathscr{A}))$ such that $\mathscr{D}(\mathscr{A}) \not\models Q\sigma\tau$, where $\mathscr{D}(\mathscr{A})$ is the diagram of $\mathscr{A}$ and $\mathscr{N}(\mathscr{A})$ is the set of new constants added to the initial signatures to describe $\mathscr{A}$.

For all t in $\mathscr{M}$, $t\tau$ is strongly $E_{BODY}(\mathscr{A})$-irreducible, since we can assume that all parameter variables on the left-hand sides of E_{BODY} are linear, as mentioned before Theorem 4.12. Let $var(Q\sigma\tau) = \{x_1, \ldots, x_k\}$ and consider a ground substitution ϕ such that $\sigma\tau\phi$ is strongly $E_{BODY}(\mathscr{A})$-irreducible and:

1. $\forall i \in [1 \ldots k], |x_i\phi| > |Q\sigma\tau|$,
2. $\forall i, j \in [1 \ldots k], i \neq j, \quad ||x_i\phi| - |x_j\phi|| > |Q\sigma\tau|$.

Note that such a substitution instance exists by using clause 2 of the definition of test set.

$\mathscr{D}(\mathscr{A}) \not\models Q\sigma\tau\phi$ since for all $i \in [1 \ldots k]$, x_i is a non-parameter variable and $x_i\phi$ is $\mathscr{D}(\mathscr{A})$-irreducible. Assume now that there exists t in $\mathscr{M}$, a rule $p \Rightarrow g \rightarrow d$ in E_{BODY} and a substitution α such that $g\alpha$ is a subterm of $t\tau\phi$. Since $\sigma\tau\phi$ is strongly

[8] If $g_j\sigma$ and $d_j\sigma$ are incomparable, then both $c_j\sigma$ and $d_j\sigma$ must be strongly E_{BODY}-irreducible.

$E_{BODY}(\mathscr{A})$-irreducible, there is a strict position u in t such that $t\tau\phi/u$ is an instance of g. Let v be a non-variable position of g, v is a non-variable position of $t\tau/u$. Otherwise, there are two cases to consider:

1. if $sdepth(E_{BODY}) < depth(E_{BODY})$ and E_{BODY} is left-linear, then we have $|v| > D(PS)$, which implies that $|v| \geqslant depth(E_{BODY})$. Now, since $sdepth(E_{BODY}) < depth(E_{BODY})$ there is a rule whose left-hand side g' satisfies $depth(g') > |v| \geqslant depth(E_{BODY})$ and $depth(g') \leqslant depth(E_{BODY})$, which is absurd.

2. otherwise, we have $|v| > D(PS) = depth(E_{BODY})$ and $|v| \leqslant depth(E_{BODY})$, which is absurd.

So necessarily v is a non-variable position of $t\tau/u$, Now, we reason as in the proof of Theorem 4.12. We conclude that $t\tau$ contains an instance of g, which is absurd since $t\tau$ is strongly $E_{BODY}(\mathscr{A})$-irreducible.

Therefore, $E_{BODY}(\mathscr{A}) \not\models_{ind} Q\sigma\tau\phi$ since $\rightarrow_{E_{BODY}(\mathscr{A})}$ is ground convergent for all models $\mathscr{A}$ of E_{PAR}. Thus, C is not an inductive theorem of PS.

It is possible to compute test sets for equational theories (see [26, 11, 29, 23]). Unfortunately, no algorithm exists for the general case of conditional theories. This is because of the fact that such a computation requires some kind of induction. However, in [30, 8] some methods are described for computing test sets for conditional specifications over a *free set of constructors*.

The next proposition is analogous to one from [8] and permits us to compute a test set in a parameterized conditional specification if the specification is sufficiently complete and the constructors are free.

Proposition 5.7. *Let $PS = (PAR, BODY)$ be a sufficiently complete parameterized specification over free constructors such that $\rightarrow_{E_{BODY}(\mathscr{A})}$ is noetherian.*[9] *The set $\mathscr{T}$ of constructor terms (up to variable renaming) of depth $\leqslant D(PS)$ where the variables of non-parameter and non-nullary sort may occur only at depth $D(PS)$ is a test set for PS.*

Proof. Let $\mathscr{A}$ be a model of E_{PAR} and t in $T(F(\mathscr{A}))$. As PS is sufficiently complete, there exists t' in $T(C_{BODY}(\mathscr{A}))$ such that $t \rightarrow^*_{E_{BODY}(\mathscr{A})} t'$. On the other hand, $\rightarrow_{E_{BODY}(\mathscr{A})}$ is noetherian and for each conditional rule $p \Rightarrow l \rightarrow r \in E_{BODY}$, if $l \in T(C_{BODY}, X)$, then $r \in T(C_{BODY}, X)$. Therefore, there exists $t'' \in T(C_{BODY}(\mathscr{A}))$ such that $t' \rightarrow^*_{E_{BODY}(\mathscr{A})} t''$ and t'' is irreducible by $E_{BODY}(\mathscr{A})$. This implies that $t \rightarrow^*_{E_{BODY}(\mathscr{A})} t''$. So any irreducible term in $T(F(\mathscr{A}))$ is built only with constructors and elements of $\mathscr{N}(\mathscr{A})$ and therefore is an instance of an element of $\mathscr{T}$. By construction, any non-ground term in $\mathscr{T}$ has non-parameter variables at depth greater than or equal to $D(PS)$. Since the constructors are free, any variable of non-parameter and non-nullary sort may be replaced by infinitely many different constructor terms. Therefore, any non-ground term in $\mathscr{T}$ is infinitary.

[9] To guarantee that $\rightarrow_{E_{BODY}(\mathscr{A})}$ is noetherian, it is sufficient to assume that $\rightarrow_{E_{BODY}}$ is noetherian and no left-hand side of an equation of E_{BODY} contains a symbol from F_{PAR}.

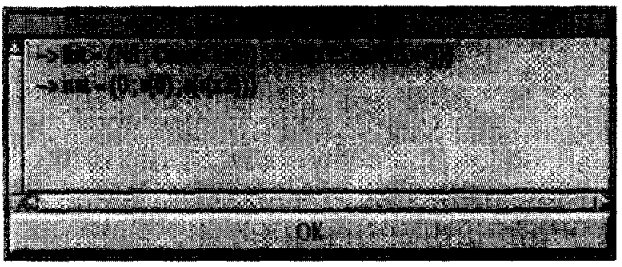

Fig. 8. Test set.

Example 5.8 (*Example 3.1 continued*). The output of the SPIKE procedure that computes a test set is given in Fig. 8.

6. An inductive procedure for parameterized specifications

6.1. Inference rules

Our procedure is defined by a set of transition rules (see Fig. 9) which are applied to pairs (E,H), where E is the set of conjectures and H is the set of inductive hypotheses.

The *generate*[10] rule allows us to derive lemmas and initiates induction steps. The *case simplify* rule simplifies a conjecture with conditional rules where the disjunction of all conditions is inductively valid. The *simplify* rule reduces a clause C with axioms from $E_{BODY} \cup E_{PAR}$, induction hypotheses from H, and other conjectures which are not yet proved. Note that *simplify* permits mutual simplification of conjectures. This rule implements *simultaneous induction* and is crucial for efficiency. The *subsumption* rule deletes clauses C subsumed by an element of $E_{BODY} \cup E_{PAR} \cup H \cup E$. The role of *deletion* is obvious. The *disproof* rule is applied if a quasi-inconsistent clause is detected. The *fail* rule is applied to (E,H) if no other rule can be applied to $C \in E$. An I-derivation is a sequence of states:

$$(E_0,\emptyset) \vdash_I (E_1,H_1) \vdash_I \cdots \vdash_I (E_n,H_n) \vdash_I \cdots$$

An I-derivation fails if it terminates with the rule *fail* or *disproof*.

6.2. Soundness

The soundness of the procedure based on our inference system relies on a fairness assumption: every conjecture to be checked must be considered at some step. More formally, a derivation $(E_0,\emptyset) \vdash_I (E_1,H_1) \vdash_I \cdots$ is *fair* if either it fails or it is infinite and the set of persisting clauses $(\bigcup_{i\geqslant 0} \bigcap_{j\geqslant i} E_j)$ is empty. Then we reason by contradiction: if a non-valid clause is generated in an unfailing derivation then a minimal

[10] Let R' be a set of clauses and suppose that R is the set of all conditional rules of R'. By abuse of notation, the relation $\rightarrow_R$ will be denoted by $\rightarrow_{R'}$.

generate: $(E \cup \{C\}, H) \vdash_I (E \cup (\cup_\sigma E_\sigma), H \cup \{C\})$

if $C \equiv p \Rightarrow q$ and for every test substitution σ of C:
if $E_{PAR} \models q\sigma$, then $E_\sigma = \emptyset$;
if $C\sigma \rightarrow_{E_{BODY} \cup E_{PAR}} C'$, then $E_\sigma = \{C'\}$;
otherwise, $E_\sigma = case\_rewriting(C\sigma)$.

case simplify: $(E \cup \{C\}, H) \vdash_I (E \cup E', H)$

if E' = case_rewriting(C).

simplify: $(E \cup \{C\}, H) \vdash_I (E \cup \{C'\}, H)$

if $C \rightarrow_{E_{BODY} \cup E_{PAR} \cup H \cup E} C'$
and for each instance $S\tau$ of clauses of H used in the simplification, we have $S\tau \preceq_c C$, while for each instance $S'\theta$ of clauses of E, we have $S'\theta \prec_c C$.

subsumption: $(E \cup \{C\}, H) \vdash_I (E, H)$

if C is subsumed by another clause of $E_{BODY} \cup E_{PAR} \cup H \cup E$.

delete: $(E \cup \{C\}, H) \vdash_I (E, H)$

if C is a tautology.

disproof: $(E \cup \{C\}, H) \vdash_I Disproof$

if C is quasi-inconsistent.

fail: $(E \cup \{C\}, H) \vdash_I \square$

if no condition of the previous rules hold for C.

Fig. 9. Inference system I.

one is generated too. We show that no inference step can apply to this clause. In other words, this clause persists in the derivation. This contradicts the fairness hypothesis. Therefore, we obtain the following result.

Theorem 6.1. *Let PS be a parameterized specification and let* $(E_0, \emptyset) \vdash_I (E_1, H_1) \vdash_I \cdots$ *be a fair I-derivation. If it does not fail then* $PS \models_{ind} E_0$.

Proof. We reason by absurdity. Suppose that $PS \not\models_{ind} E_0$ and let C' be a minimal element w.r.t. $\prec_c$ of the set $\{D\sigma \mid D \in \cup_i E_i$ and there is a model $\mathscr{A}$ of E_{PAR} and a ground substitution σ over $T(F(\mathscr{A}))$ that is irreducible by $E_{BODY}(\mathscr{A})$ such that $PS \not\models_{ind} D\sigma\}$. C' exists since $PS \not\models_{ind} E_0$ and $\prec_c$ is well-founded. Then, there exists a clause $C \in \cup_i E_i$ minimum w.r.t. the subsumption ordering, and a ground substitution σ such that $C' \equiv C\sigma$. It is sufficient to prove that C cannot be simplified or deleted, and that *generate* cannot be applied to C; this shows that *fail* or *disproof* applies since the clause C must not persist in the derivation by the fairness hypothesis. Hence, let us assume that $C \in E_j$ and $(E_j, H_j) \vdash_I (E_{j+1}, H_{j+1})$ by some rule applied to C.

We discuss now the situation according to which rule is applied. In every case we shall derive a contradiction. In order to simplify the notations we write E for E_j and H for H_j.

generate: Suppose that $(E_j, H_j) \vdash_I (E_{j+1}, H_{j+1})$ by *generate* on $C \equiv p \Rightarrow q$. Since σ is a ground substitution over $T(F(\mathscr{A}))$ that is irreducible by $E_{BODY}(\mathscr{A})$, there exists a test substitution σ_0 of C and a substitution θ such that $\sigma \equiv \sigma_0\theta$. $E_{PAR} \not\models q\sigma_0$ since $PS \not\models_{ind} C\sigma_0$; we have two possibilities:

1. If there exists a clause C' such that $C\sigma_0 \rightarrow_{E_{BODY} \cup E_{PAR}} C'$ then $C\sigma \rightarrow_{E_{BODY} \cup E_{PAR}} C'\theta$. Therefore, $PS \not\models_{ind} C'\theta$. On the other hand, $C'\theta \prec_c C\sigma$ and $C' \in \bigcup_i E_i$. This shows a contradiction since it proves that we can find an instance of a clause in $\bigcup_i E_i$ that is not valid and that is smaller than $C\sigma$ with respect to $\prec_c$.

2. Assume that the rule *case_rewriting* is applied to $C\sigma$. Then, consider all the rules: $p_1 \Rightarrow t_1 \rightarrow r_1$, $p_2 \Rightarrow t_2 \rightarrow r_2, \ldots, p_n \Rightarrow t_n \rightarrow r_n$ in E_{BODY} such that there exists a sequence of positions $u_1, u_2, \ldots, u_n$ in $C\sigma$ and $C\sigma/u_1 \equiv t_1\sigma_1$, $C\sigma/u_2 \equiv t_2\sigma_2, \ldots, C\sigma/u_n \equiv t_n\sigma_n$ and $p_1\sigma_1 \vee p_2\sigma_2 \vee \cdots \vee p_n\sigma_n$ is an inductive theorem of PS. Hence, the result of the application of *case_rewriting* is

$$\{p_1\sigma_1 \Rightarrow C\sigma[r_1\sigma_1]_{u_1}, \ldots, p_n\sigma_n \Rightarrow C\sigma[r_n\sigma_n]_{u_n}\}$$

Then there exists k such that $PS \models_{ind} p_k\sigma_k$. Let $C' \equiv p_k\sigma_k \Rightarrow C\sigma[r_k\sigma_k]_{u_k}$; we have $PS \models_{ind} (p_k \Rightarrow t_k \rightarrow r_k)\sigma_k$. Therefore, $PS \models_{ind} r_k\sigma_k = t_k\sigma_k$. Putting everything together, we get $PS \not\models_{ind} C'\theta$. On the other hand, $C' \in \bigcup_i E_i$ and $C'\theta \prec_c C\sigma$, this is also absurd.

case simplify: This case is similar to the previous one.

simplify: Suppose that the *simplify* rule applies to C, then, there exists a clause C' such that $C \rightarrow_{E_{BODY} \cup E_{PAR} \cup H \cup E} C'$, then $C\sigma \rightarrow_{E_{BODY} \cup E_{PAR} \cup H \cup E} C'\sigma$. For each instance $S\tau$ of clauses of $H \cup E$ used in the rewriting step, we have $S\tau \prec_c C\sigma$ (we cannot have $S\tau \approx_c C\sigma$ and $S \in H$, otherwise, the *generate* rule has been applied to S. Therefore, *generate* can be also applied to C in contradiction with a previous case). Then, we have $PS \models_{ind} S\tau$. Therefore, $PS \not\models_{ind} C'\sigma$. On the other hand, $C'\sigma \prec_c C\sigma$ and $C' \in \bigcup_i E_i$, which is absurd.

subsumption: Since $PS \not\models_{ind} C\sigma$, C cannot be subsumed by a clause of $E_{BODY} \cup E_{PAR}$. If there is $C' \in H \cup (E \setminus \{C\})$ such that $C \equiv C'\tau \vee r$, we have $PS \not\models_{ind} C'\tau\sigma$, then $C \equiv C'$ since C is minimum in $\bigcup_i E_i$ w.r.t. the subsumption ordering. As a consequence $C' \notin (E \setminus \{C\})$. On the other hand, $C' \notin H$. Otherwise, the *generate* rule has been applied to C'. Therefore, *generate* can be also applied to C in contradiction with a previous case. Hence, this rule cannot be applied to C.

delete: Since $PS \not\models_{ind} C\sigma$, C cannot be a tautology and this rule need not be considered.

Since every I-derivation from $(E,\emptyset)$ to $(\emptyset,H)$, where H is some set of clauses, is fair the conjectures of E are inductive theorems of PS. This remark is important from a practical point of view. Note also that E is valid even when the derivation is infinite.

If *disproof* is applied at step k, then a quasi-inconsistent clause is detected and therefore, from Theorem 5.6, we conclude that some conjecture in E_k is false, if for every model $\mathscr{A}$ of E_{PAR} $\rightarrow_{E_{BODY}(\mathscr{A})}$ is ground convergent over $T(F(\mathscr{A}))$ and all parameter variables on the left-hand sides of E_{BODY} are linear. The initial conjectures of E_0 are not inductive theorems of PS either. This is a consequence of the next result.

Lemma 6.2. *Let* $(E_0,\emptyset) \vdash_I (E_1,H_1) \vdash_I \cdots$ *be an I-derivation. If for all* i *such that* $i \leqslant j$ *we have* $PS \models_{ind} E_i$ *then* $PS \models_{ind} E_{j+1}$.

Proof. Suppose that $(E_j,H_j) \vdash_I (E_{j+1},H_{j+1})$ by *generate* on C. Let σ be a test substitution of C. If $E_{PAR} \not\models C\sigma$, then there are two cases to consider: (i) if there exists C' such that $C\sigma \rightarrow_{E_{BODY} \cup E_{PAR}} C'$. Then, we have $PS \models_{ind} C'$. (ii) otherwise, there exists a sequence of conditional rules $p_1 \Rightarrow t_1 \rightarrow r_1$, $p_2 \Rightarrow t_2 \rightarrow r_2, \ldots, p_n \Rightarrow t_n \rightarrow r_n$ in E_{BODY} and a sequence of positions $u_1, u_2, \ldots, u_n$ in $C\sigma$ such that $C\sigma/u_1 \equiv t_1\tau_1$, $C\sigma/u_2 \equiv t_2\tau_2, \ldots, C\sigma/u_n \equiv t_n\tau_n$ and $p_1\tau_1 \vee p_2\tau_2 \vee \ldots \vee p_n\tau_n$ is an inductive theorem of PS. Assume that there exists k such that: $PS \not\models_{ind} C_k \equiv p_k\tau_k \Rightarrow C\sigma[r_k\tau_k]_{u_k}$. In other words there is a ground instance $C_k\theta$ over $T(F(\mathscr{A}))$ (without loss of generality, we can assume that $C\sigma\theta$ is ground) such that: $PS \not\models_{ind} C_k\theta$, then $PS \models_{ind} p_k\tau_k\theta$ and $PS \not\models_{ind} C\sigma\theta[r_k\tau_k\theta]$. Therefore, $PS \models_{ind} t_k\tau_k\theta = r_k\tau_k\theta$. This implies that $PS \not\models_{ind} C\sigma\theta[t_k\tau_k\theta]$, which is absurd. For *case simplify* the argument is the same as above. If $(E_j,H_j) \vdash_I (E_{j+1},H_{j+1})$ by *simplify*, then the equations which are used for simplification occur in some E_k $(k \leqslant j)$ and therefore are inductively valid by the hypothesis. Hence, E_{j+1} is inductively valid too.

The next theorem is a straightforward consequence of the above results.

Theorem 6.3. *Given a parameterized specification* $PS = (PAR, BODY)$ *such that* $\rightarrow_{E_{BODY}(\mathscr{A})}$ *is ground convergent for every model* $\mathscr{A}$ *of* E_{PAR}. *Let* $(E_0,\emptyset) \vdash_I (E_1,H_1) \vdash_I \cdots$ *be an I-derivation. If there exists* j *such that* disproof *applies to* (E_j,H_j) *then* $PS \not\models_{ind} E_0$.

6.3. Refutational completeness for parameterized boolean specifications

In this section, we shall consider axioms that are conditional rules with boolean preconditions. To be more specific, we assume there exists a sort *bool* with two free constructors $\{True, False\}$. Every rule in E_{BODY} is of type: $\bigwedge_{i=1}^{n} p_i = p_i' \Rightarrow s \rightarrow t$ where for all i in $[1 \ldots n]$, $p_i' \in \{True, False\}$. For $\alpha \in \{True, False\}$ we denote by $\overline{\alpha}$ the complementary *bool* symbol of α. Conjectures will be *boolean clauses*, i.e. clauses whose negative literals are of type $\neg(p = p')$ where $p' \in \{True, False\}$. Let f be a function symbol in D_{BODY}. If for all the rules in E_{BODY} of the form $p_i \Rightarrow f(\vec{t}_i) \rightarrow r_i$

whose left-hand sides are identical up to a renaming μ_i, we have $PS \models_{ind} \bigvee_i p_i\mu_i$, then f is called weakly complete w.r.t. PS. We say that PS is weakly complete if any function in D_{BODY} is weakly complete w.r.t. PS. Note that a *weakly complete* specification is not necessary *sufficiently complete.*

Example 6.4 (*Example 3.1 continued*). Suppose that all the rules for defining *sorted* are removed except for the rules:

$$x \leqslant y = False \Rightarrow sorted(Cons(x, Cons(y,z))) \rightarrow False$$
$$x \leqslant y = True \Rightarrow sorted(Cons(x, Cons(y,z))) \rightarrow sorted(Cons(y,z))$$

The function *sorted* is weakly complete, but it is not sufficiently complete.

Now, we can define a new inference system J from I by adding the complement rule (see Fig. 10) which transforms negative clauses to positive clauses that are easier to refute. We also remove the *fail* rule and reformulate *disproof* as in Fig. 10.

Let us assume that E_0 only contains boolean clauses. The only rule that permits us to introduce negative clauses is *case_rewriting*. Since the axioms have boolean preconditions, all the clauses generated in a J-derivation are boolean. If *disproof* is applied in a J-derivation, then there exists a positive clause C such that *generate* cannot be applied to C. Therefore there exists a test substitution σ such that $E_{PAR} \not\models C\sigma$. Moreover, $C\sigma$ does not match any left-hand side of E_{BODY}. Otherwise, the *conditional rewriting* or the *case rewriting* rule can be applied to $C\sigma$ since PS is weakly complete. As a consequence, C is a quasi-inconsistent clause. So, the new inference system J can be proved refutationally complete for boolean clauses. Thus, formally, we have the following result.

Theorem 6.5. *Given a weakly complete parameterized specification* $PS = (PAR, BODY)$ *such that* $\rightarrow_{E_{BODY}(\mathscr{A})}$ *is ground convergent for every model* $\mathscr{A}$ *of* E_{PAR}. *We assume that* E_0 *contains only boolean clauses. If a derivation issued from* $(E_0, \emptyset)$ *terminates by application of the rule* disproof, *then* $PS \not\models_{ind} E_0$. *Conversely, if* $PS \not\models_{ind} E_0$, *then all fair derivations issued from* $(E_0, \emptyset)$ *terminate by application of the rule* disproof.

complement: $(E \cup \{\neg(a = \alpha) \vee r\}, H) \vdash_J (E \cup \{(a = \overline{\alpha}) \vee r\}, H)$
if $\alpha \in \{true, false\}$.

disproof: $(E \cup \{C\}, H) \vdash_J Disproof$
if no condition of the previous rules hold for C.

Fig. 10. New rules for boolean specification.

7. Implementation and experimental results

Our implementation is based on the previous inference system. The program is able to prove the validity of a set of clauses[11] in parameterized conditional specifications. Here is an overview of the algorithm. The main data structures are: the list E_{BODY} of axioms, that are conditional rules built with the constructor discipline, the list E of conjectures (clauses) to be checked, the list E_{PAR} of parameter constraints, that are equational clauses over F_{PAR} and finally, the set H of induction hypotheses (initialized by $\emptyset$). The first step in a proof session is to check if the rules are oriented and if all defined functions are completely defined. The second step is to compute a test set for PS and also induction positions. After these preliminary tasks, the proof starts.

7.1. Results and comparison

Consider Example 3.1 and let us prove the following conjectures:

$$sorted(insert(x, y)) = sorted(y) \tag{1}$$

$$length(insert(x, y)) = s(length(y)) \tag{2}$$

$$count(x, insert(y, z)) = count(x, Cons(y, z)) \tag{3}$$

SPIKE can prove these conjectures in a completely automatic way, using 137 steps. Note that three lemmas (generated automatically) are sufficient to prove the initial conjectures (see Fig. 11).

By assuming the conjectures (1)–(3) as lemmas, SPIKE can easily prove the following conjectures in a completely automatic way:

$$sorted(isort(x)) = True$$

$$length(isort(x)) = length(x)$$

$$count(x, isort(y)) = count(x, y)$$

$$dif(x, y) = False \vee count(x, insert(y, insert(x, z))) = s(count(x, z))$$

Now consider the same example with lists of natural numbers, using the method of [8], we have the following test sets and induction positions (see Fig. 12). To prove conjectures (1) and (2) without parameters, SPIKE used 239 steps. In addition, 14 lemmas were generated automatically during the proof. On the other hand, the proof of conjecture (3) diverges. This example illustrates that with parameterized specifications we have a smaller test set and fewer induction positions, permitting us to obtain shorter and structured proofs.

[11] In our SPIKE system, we use the sequent style notation.

7.2. Refutation of conjectures

The parameterized specification PS of Example 3.1 is weakly complete and the parameter variables on the left-hand sides of E_{BODY} are linear. On the other hand, using the method of [3], we can easily prove that $\rightarrow_{E_{BODY}(\mathscr{A})}$ is ground convergent

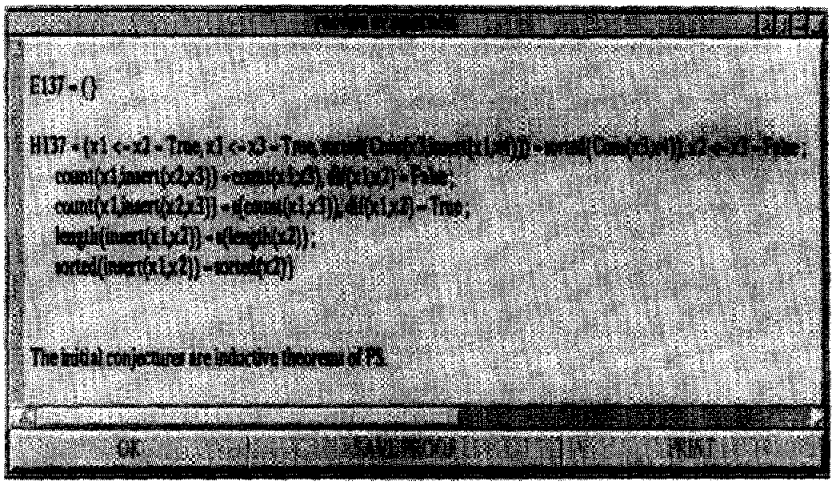

Fig. 11. Success.

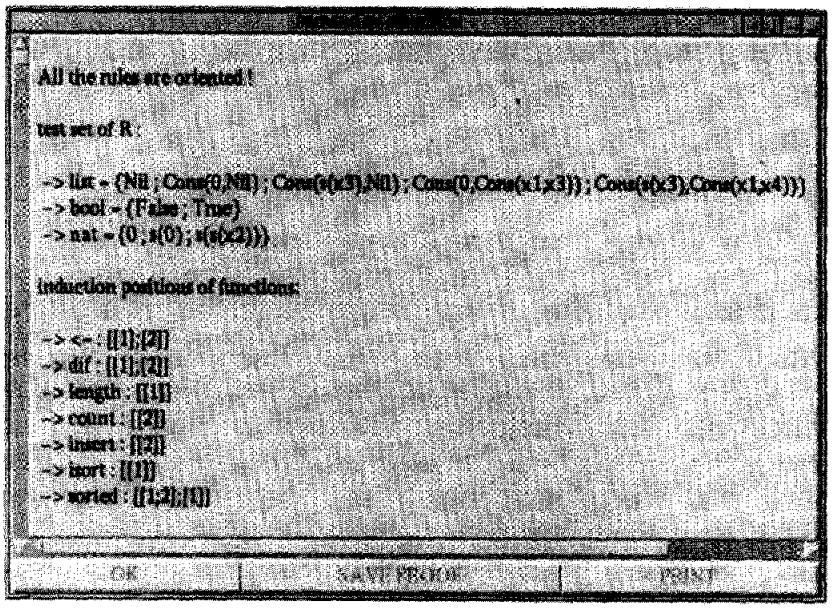

Fig. 12. Test Set and induction positions (without parameters).

for every model $\mathscr{A}$ of E_{PAR}. Under these hypotheses, SPIKE can refute any false conjectures in finite time. Here is an example of a refutation of a false conjecture:

The conjecture to be disproved is

$$x \leqslant y = False \vee sorted(Cons(x, l)) = False \vee sorted(insert(x, Cons(y, l))) = True \quad (4)$$

Conjecture (4) is simplified by E_{BODY}, using the *simplify* rule into

$$x \leqslant y = False \vee sorted(Cons(x, l)) = False \vee sorted(Cons(y, l)) = True \quad (5)$$

In order to prove subgoal (5), we apply the *generate* rule with the following *test substitution*:

$$l \leftarrow \{Nil; Cons(x1, Nil); Cons(x1, Cons(x3, x4))\}$$

Among the generated clauses, we obtain

$$x \leqslant z = True \Rightarrow x \leqslant y = False \vee sorted(Cons(y, Cons(z, Nil))) = True \vee sorted(Cons(z, Nil)) = False \quad (6)$$

Since in our framework an atom is considered to be simpler than its negation, we simplify clauses into positive ones. The simplification of subgoal (6), using the *complement* rule, gives us

$$x \leqslant z = False \vee x \leqslant y = False \vee sorted(Cons(y, Cons(z, Nil))) = True \vee sorted(Cons(z, Nil)) = False \quad (7)$$

Subgoal (7) is simplified by E_{BODY}, using the *simplify* rule, into

$$x \leqslant y = False, x \leqslant z = False, sorted(Cons(y, Cons(z, Nil))) = True, True = False \quad (8)$$

Since we have $\neg(True = False)$ as a constraint, subgoal (8) is simplified [12] into

$$x \leqslant y = False, x \leqslant z = False, sorted(Cons(y, Cons(z, Nil))) = True \quad (9)$$

In order to prove subgoal (9), we apply *case simply* using E_{BODY}, we obtain

$$y \leqslant z = False \Rightarrow x \leqslant y = False, x \leqslant z = False, False = True \quad (10)$$

$$y \leqslant z = True \Rightarrow x \leqslant y = False, x \leqslant z = False, sorted(Cons(z, Nil)) = True$$

The simplification of subgoal (10), using the *complement* rule, gives us

$$x \leqslant y = False, x \leqslant z = False, False = True, y \leqslant z = True \quad (11)$$

[12] We can easily prove the soundness of this simplification rule.

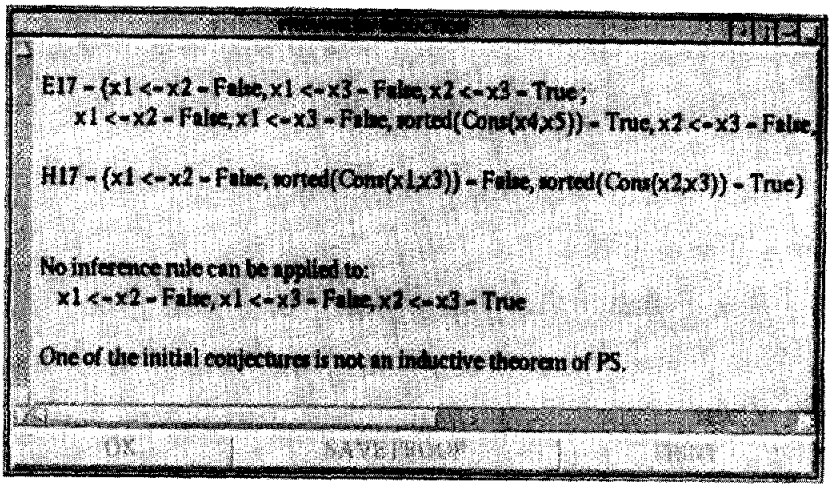

Fig. 13. Refutation of a conjecture.

Subgoal (11) is simplified by E_{PAR} into

$$x \leqslant y = False, x \leqslant z = False, y \leqslant z = True \tag{12}$$

No inference rule can be applied to subgoal (12) and therefore it is quasi-inconsistent (see Example 5.5). We conclude that conjecture (4) is not an inductive theorem of *PS* (see Fig. 13).

8. Conclusion

We have proposed a new procedure for proof by induction in parameterized conditional specifications. Unlike our previous procedure [8], this new one, when limited to non-parameterized conditional specifications, can refute general clauses; refutational completeness is also preserved for boolean ground convergent rewrite systems even if the functions are not sufficiently complete and the constructors are not free. The property of sufficient completeness is very important in specification systems but is in general undecidable. We have given a procedure for testing this property for parameterized conditional specifications.

The method is implemented in the prover SPIKE. This system has proved interesting examples in a completely automatic way [4], that is, without interaction with the user and without ad hoc heuristics. Experiments illustrate the improvements in length and structure of proofs, due to parameterization. Unlike the well-known induction provers, SPIKE guarantees when it fails that one of the initial conjectures is not an inductive theorem provided that the axioms are boolean and ground convergent. Moreover, SPIKE offers facilities to check and complete specifications.

We plan to generalize the method to get refutational completeness for a larger class of rewrite systems. Another powerful extension is to allow for generalization techniques, such as in the traditional induction method. How this can be done and the possible implications with respect to soundness and refutational completeness, still have to be studied very carefully.

Acknowledgements

I am grateful to Michaël Rusinowitch, Hélène Kirchner, Toby Walsh, Peter Padawitz and Jürgen Avenhaus for many fruitful discussions. I would like to thank Christopher Lynch and Pierre Lescanne for comments on earlier drafts. This work is partially supported by the Esprit Basic Research working group 6112, COMPASS.

References

[1] R. Aubin, Mechanizing structural induction, *Theoret. Comput. Sci.* **9** (1979) 329–362.
[2] L. Bachmair, Proof by consistency in equational theories, in: *Proc. 3rd IEEE Symp. on Logic in Computer Science*, Edinburgh, UK (1988) 228–233.
[3] K. Becker, Inductive proofs in specifications parameterized by a built-in theory, SEKI-Report SR-92-02, Universität Kaiserslautern, 1992.
[4] A. Bouhoula, Preuves automatiques par récurrence dans les théories conditionnelles, Ph.D. Thesis, Université de Nancy 1, March 1994.
[5] A. Bouhoula, Spike: a system for sufficient completeness and parameterized inductive proof, in: A. Bundy, ed., *Proc. 12th Internat. Conf. on Automated Deduction*, Nancy, France, Lecture Notes in Artificial Intelligence, Vol. 814 (Springer, Berlin, 1994) 836–840.
[6] A. Bouhoula, Sufficient completeness and parameterized proofs by induction, in: *Proc. 4th Internat. Conf. on Algebraic and Logic Programming*, Madrid, Spain, Lecture Notes in Computer Science, Vol. 850 (Springer, Berlin, 1994) 23–40.
[7] A. Bouhoula and M. Rusinowitch, Automatic case analysis in proof by induction, in: R. Bajcsy, ed., *Proc. 13th Internat. Joint Conf. on Artificial Intelligence*, Chambéry, France, Vol. 1 (Morgan Kaufmann, LOS Altos, CA, 1993) 88–94.
[8] A. Bouhoula and M. Rusinowitch, Implicit induction in conditional theories, *J. Automated Reasoning* **14** (1995) 189–235.
[9] A. Bouhoula, E. Kounalis and M. Rusinowitch, Automated mathematical induction, *J. Logic Comput.* **5** (1995) 631–668.
[10] R.S. Boyer and J.S. Moore, *A Computational Logic Handbook* (Academic Press, New York, 1988).
[11] R. Bündgen and W. Küchlin, Computing ground reducibility and inductively complete positions, in: N. Dershowitz, ed., *Proc. 3rd Conf. on Rewriting Techniques and Applications*, Chapel Hill, NC, Lecture Notes in Computer Science, Vol. 355 (Springer, Berlin, 1989) 59–75.
[12] A. Bundy, F. van Harmelen, C. Horn and A. Smaill, Rippling: a heuristic for guiding inductive proofs, *Artificial Intelligence* **62** (1993) 183–253.
[13] H. Comon, Sufficient completeness, term rewriting systems and "anti-unification", in: *Proc. 8th Internat. Conf. on Automated Deduction*, Oxford, England, July 1986.
[14] N. Dershowitz, Well-founded orderings, ATR-83-8478-3, Information Science Research Office, The Aerospace Corporation, El Segundo, CA, 1983.
[15] N. Dershowitz, Termination of rewriting, *J. Symbolic Comput.* **3** (1987) 69–116.
[16] N. Dershowitz and J.-P. Jouannaud, Rewrite systems, in: J. van Leeuwen, ed., *Handbook of Theoretical Computer Science* (Elsevier, Amsterdam, 1990).
[17] H. Ehrig and B. Mahr, *Fundamentals of Algebraic Specification 1. Equations and Initial Semantics*, EATCS Monographs on Theoretical Computer Science Vol. 6 (Springer, Berlin, 1985).

[18] L. Fribourg, A strong restriction of the inductive completion procedure, *J. Symbolic Comput.* **8** (1989) 253–276.

[19] H. Ganzinger, Ground term confluence in parametric conditional equational specifications, in: F.J. Brandenburg, G. Vidal-Naquet and M. Wirsing, eds., *Proc. STACS '87, Lecture Notes in Computer Science*, Vol. 247 (Springer, Berlin, 1987) 286–298.

[20] B. Gramlich, UNICOM: a refined completion based inductive theoremprover, in: M.E. Stickel, ed., *Proc. 10th Internat. Conf. on Automated Deduction*, Lecture Notes in Artificial Intelligence, Vol. 449 (Springer, Berlin, 1989) 655–656.

[21] J.V. Guttag, Abstract data types and software validation, *Commun. ACM* **21** (1978) 1048–1064.

[22] J.V. Guttag and J.J. Horning, The algebraic specification of abstract data types, *Acta Inform.* **10** (1978) 27–52.

[23] D. Hofbauer and M. Huber, Linearizing term rewriting systems using test set, *J. Symbolic Comput.* **17** (1994) 91–129.

[24] G. Huet and J.-M. Hullot, Proofs by induction in equational theories with constructors, *J. Comput. System Sci.* **25** (1982) 239–266. Preliminary version in *Proc. 21st Symp. on Foundations of Computer Science* (IEEE, New York, 1980).

[25] J.-P. Jouannaud and E. Kounalis, Automatic proofs by induction in theories without constructors, *Inform. and Comput.* **82** (1989) 1–33.

[26] D. Kapur, P. Narendran and H. Zhang, On sufficient completeness and related properties of term rewriting systems, *Acta Inform.* **24** (1987) 395–415.

[27] H. Kirchner, A general inductive completion algorithm and application to abstract data types, in: R. Shostak, ed., *Proc. 7th Internat. Conf. on Automated Deduction*, Napa Valley, CA, Lecture Notes in Computer Science, Vol. 170 (Springer, Berlin, 1984) 282–302.

[28] E. Kounalis, Completeness in data type specifications, in: B. Buchberger, ed., *Proc. EUROCAL Conf.*, Linz, Austria, Lecture Notes in Computer Science, Vol. 204 (Springer, Berlin, 1985) 348–362.

[29] E. Kounalis, Testing for the ground (co-)reducibility property in term-rewriting systems, *Theoret. Comput. Sci.* **106** (1992) 87–117.

[30] E. Kounalis and M. Rusinowitch, Mechanizing inductive reasoning, in: *Proc. American Association for Artificial Intelligence Conf.*, Boston (AAAI Press and MIT Press, New York and Cambridge, 1990) 240–245.

[31] W. Küchlin, Inductive completion by ground proof transformation, in: H. Aït-Kaci and M. Nivat, ed., *Coll. on the Resolution of Equations in Algebraic Structures, Vol. 2: Rewriting Techniques* (Academic Press, New York, 1989) 211–244.

[32] A. Lazrek, P. Lescanne and J.-J. Thiel, Tools for proving inductive equalities, relative completeness and ω-completeness, *Inform. and Comput.* **84** (1990) 47–70.

[33] D.R. Musser, On proving inductive properties of abstract data types, in: *Proc. 7th ACM Symp. on Principles of Programming Languages* (ACM, New York, 1980) 154–162.

[34] M. Navarro and F. Orejas, Parameterized Horn clause specifications: proof theory and correctness, in: TAPSOFT'87, Pisa, 1987.

[35] P Padawitz, Towards a proof theory of parameterized specifications, *Semantics Data Types* **173** (1985) 375–391.

[36] P. Padawitz, Parameter-preserving data type specifications, *J. Comput. System Sci.* **34** (1987) 179–209.

[37] U.S. Reddy, Term rewriting induction, in: M.E. Stickel, ed., *Proc. 10th Internat. Conf. on Automated Deduction*, Kaiserslautern, Germany, Lecture Notes in Computer Science, Vol. 449 (Springer, Berlin, 1990) 162–177.

[38] M. Wirsing, Algebraic specification, in: J. van Leeuwen, ed., *Handbook of Theoretical Computer Science B: Formal Methods and Semantics*, Ch. 13 (Elsevier, Amsterdam, 1990) 675–788.

[39] H. Zhang, D. Kapur and M.S. Krishnamoorthy, A mechanizable induction principle for equational specifications, in: E. Lusk and R. Overbeek, eds., *Proc. 9th Internat. Conf. on Automated Deduction*, Argonne, USA, Lecture Notes in Computer Science, Vol. 310 (Springer, Berlin, 1988) 162–181.